CART - Dokumentation

Ubiquitous Computing
CVD Semester VI, SS2017

Schneider, Fabian - 2141115
Schmidt, Adrian - 2140622
Quinders, Tom - 2141339

Übersicht

1. Motivation

CART ist ein selbstständiges Kameraführungssystem. Es verfolgt Carrera-Wagen mit einem auf dem Dach befestigten Infrarotlicht automatisch über eine bestimmte Strecke. Dabei versucht es die Wagen stets in der Mitte des Kamerabildes zu halten. Durch Benutzung eines Mikrocontrollers und allgemein kleinen Bauteilen ist das System sehr kompakt. Ein Kameradrehkopf soll sowohl horizontale als auch vertikale Bewegungen ermöglichen.

Dieses System könnte für die Erkennung und Verfolgung von realen Wagen auf Rennstrecken ausgebaut werden und im Motorsport so wie in der Filmindustrie Verwendung finden. Ebenso wäre es möglich, das System im Straßenverkehr für Zählungen oder Überwachungen anzuwenden.

2. Vorhandene Systeme

Ein ähnliches Konzept wie das CART-System verfolgen die Verkehrszähler. Verkehrszähler-Systeme erkennen PKWs automatisch im Verkehr und zählen diese. Diese Systeme sind dabei nicht auf die Verfolgung der Wagen ausgelegt, die dazugehörigen Kameras sind also statisch. Im Fokus liegt eher die Erhebung von Statistiken über den Straßenverkehr auf der beobachteten Strecke, zum Beispiel die Anzahl der Geschwindigkeitsüberschreitungen [1].

Ebenso benutzt das Militär ähnliche Konzepte in Drohnen und Abwehrsystemen. Hierbei können sowohl PKWs als auch Personen oder Gebäude automatisch identifiziert und verfolgt werden. In diesen Systemen ist eine hohe Auflösung von Vorteil, um Personen von einer großen Höhe aus korrekt zu identifizieren.

Für den persönlichen Gebrauch im Bereich der sportlichen Aktivitäten existieren außerdem Systeme wie die Soloshot Kamera [2]. Durch einen Sensor am Arm des Sportlers wird der Kamera gesendet, wo sich das Zielobjekt relativ zu der Kamera befindet. Die Kamera bewegt sich in Richtung des Sensors und kann dabei auch automatisch zoomen.
Dieses Konzept verlässt sich nicht auf die Bildverarbeitung. Dadurch erzielt es stabilere Ergebnisse für einzelne Ziele, ist aber stark limitiert in dem was es verfolgen kann - nämlich nur die Person, die den Sensor trägt.

3. Konzept

Der Prototyp des CART Systems besteht aus einem Mikrocontroller, einer Kamera und Servomotoren für die Bewegung der Kamera. Der Mikrocontroller agiert als Zentrale Recheneinheit, die die Bilder der Kamera verarbeitet und an vordefinierten Features sein Zielobjekt erkennt. Mit der erkannten Position des Zielobjekts wird dann die Bewegung errechnet, die die Motoren ausführen müssen, um das Objekt im Fokus der Kamera zu behalten.

3.1. Erkennung und Kamera

Zunächst bestand die Überlegung, ob ein Carrera-Wagen in einem Bild erkannt werden könnte. Diese Möglichkeit war zwar mit Hilfe der OpenCV Bibliothek gegeben, jedoch benötigte man dafür mehr Speicherplatz und Rechenleistung als auf dem Mikrocontroller vorgesehen. Des Weiteren müsste man den Prozess zunächst auf das korrekte Aussehen der Wagen “trainieren”. Dieser Prozess würde den geplanten Zeitrahmen sprengen. Jedoch kann die Technologie mit diesem Prozess erweitert werden.

Schließlich fiel die Entscheidung auf die Benutzung von Infrarotlicht zur Verfolgung des Objektes. Hierbei sollte der Wagen mit einer Infrarot-LED ausgestattet werden. Das Infrarotlicht ist dabei nicht sichtbar für das menschliche Auge, aber die Kamera erfasst dieses. Von der Kamera soll das Infrarotlicht erkannt, identifiziert und verfolgt werden. Dadurch entsteht automatisch eine Verfolgung des Wagens.
Durch die Benutzung einer Kamera, die Infrarotlicht aufnimmt, wird das Bild natürlich gestört. Im Prototypen sollte nur eine Kamera verwendet werden. Da der Testlauf in einem abgedunkelten Raum stattfinden sollte um mögliche Fehlerquellen zu vermeiden. Möchte man das System aber erweitern und weiterhin mit Infrarotlicht-Erkennung agieren, würde man eine zweite Kamera benötigen, die das Bild ohne Infrarotlicht aufnimmt.

3.2. Verfolgung und Motoren

Für die Bewegung der Kamera war ein Mechanismus benötigt, der den Kopf horizontal und vertikal verstellen kann. Zunächst wurde überlegt, für die Bewegung der Kamera eine Art Taumelscheibe, wie sie auch bei Hubschraubern zu finden ist, zu verwenden. Nach kurzer Forschung stellte sich jedoch heraus, dass dieser Mechanismus nicht die gewünschten Bewegungen ermöglichen und schwer zu kontrollieren sein würde. Des Weiteren existierte so ein Bauteil nicht im vorgefertigten Zustand, was unsere Arbeit um einiges erschwert hätte.

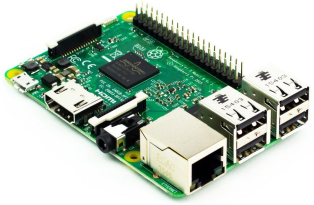
Weitere Forschung führte zu dem Entschluss, einen simplen Drehkopf zu verwenden.

4. Konstruktion

4.1. Bauteile

Folgende Bauteile wurden im Aufbau des Prototypen verwendet.

- Raspberry Pi 3
- Raspberry Pi NoIR Camera
- Raspberry Pi Kamera Verbindungskabel
- 3D-Druck eines Verbindungsstücks
- Pan-Tilt-Roll Mechanismus “Fat Shark”
- Kingbright IR-Emitter
- AA-Batterien mit Batteriehalter und Druckknopfanschluss
- Batterieclip 1x9V Block Druckknopfanschluss
- Raspberry Pi Verbindungskabel
- Isolierband
- Knopfzellen
- Kohleschicht-Widerstände
- Steckplatinen



[3] Raspberry Pi 3

[4] Fat Shark

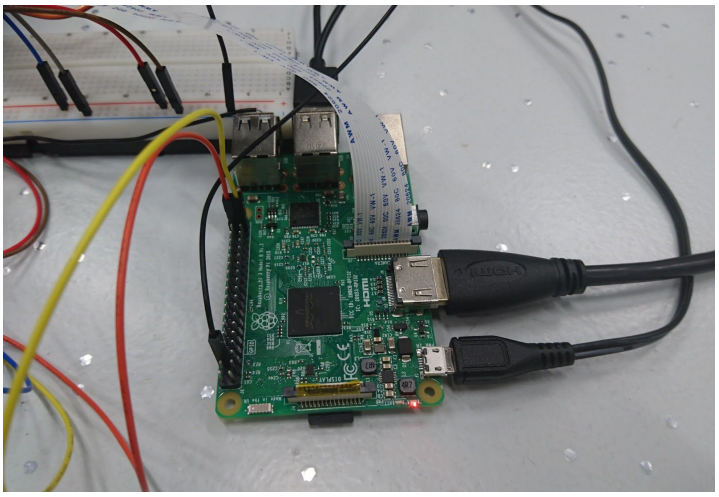
[5] NoIR Kamera

4.2. Raspberry Pi 3

Der verwendete Mikrocontroller des Prototypen ist ein Raspberry Pi 3. Auf dem Mikrocontroller liegt der Programmcode.

Die Stromzufuhr des Raspberry Pi geschieht über die Micro-USB Schnittstelle. Um den Programm-Prototypen auf dem Mikrocontroller zu starten sind Maus und Tastatur so wie ein Computerbildschirm benötigt. Diese schließt man jeweils über die herkömmlichen USB - und HDMI Ports an.

Der Raspberry Pi ist über Verbindungskabel verbunden mit einer Steckplatine, auf welcher sich der Drehkopf mit der Kamera befindet.



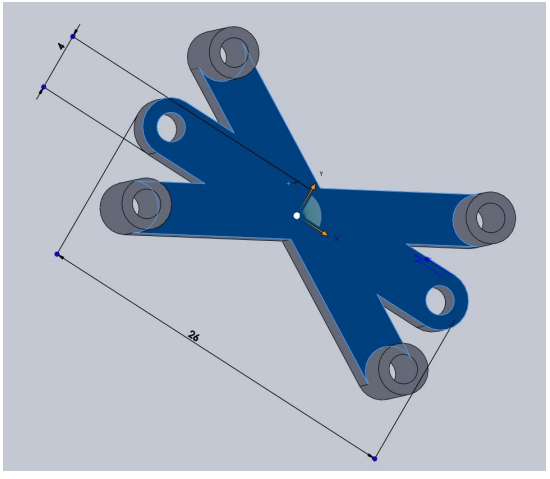
Raspberry Pi 3 in der Konstruktion

4.3. Kamera Konstruktion

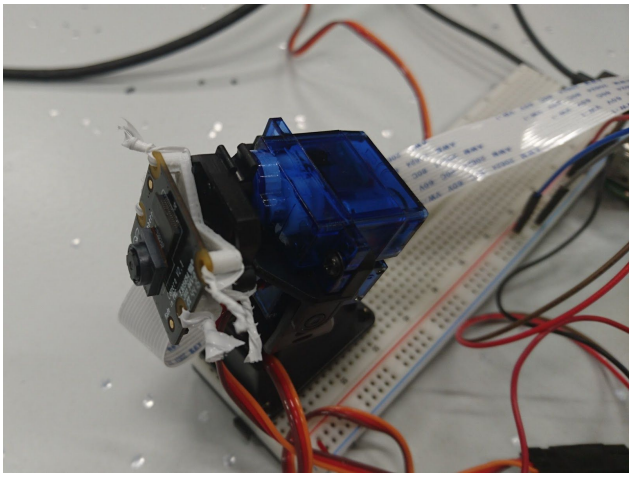
An der Steckplatine ist der “Fat Shark” Mechanismus befestigt. Dieser ist über die Steckplatine und die Verbindungskabel mit den GPIO Pins des Raspberry Pi verbunden. Die Stromzufuhr des Mechanismus’ geschieht über die AA Batterien im Batteriehalter, welcher über den Druckknopfanschluss ebenfalls mit der Steckplatine verbunden ist.

Damit die Kamera am Kopf befestigt werden kann, befindet sich zwischen den beiden Teilen das eigens erstellte und 3D-gedruckte Verbindungsstück. Dieses schafft einen Abstand zwischen der Kamera und dem Kopf, ohne den die Kamera nicht korrekt auf das Konstrukt passen würde. Die Teile wurden provisorisch mit verdrillten Drähten befestigt.

Die Kamera ist über das Camera Serial Interface (CSI) des Raspberry Pi durch das Kamerakabel mit dem Mikrocontroller verbunden.



Verbindungsstück in Solidworks

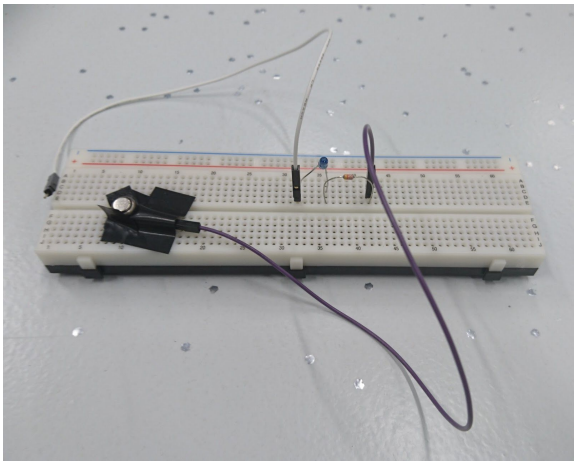


Kamerakonstruktion

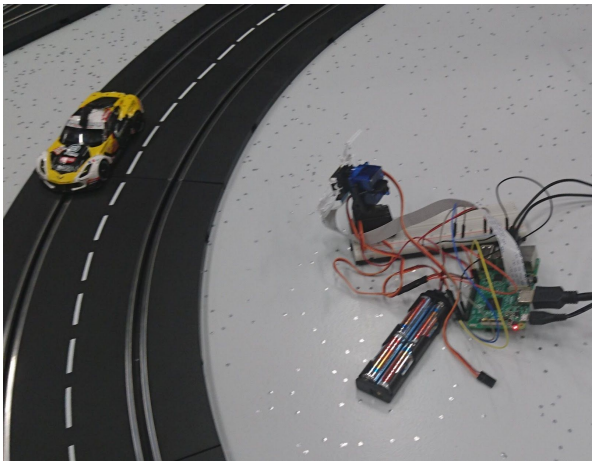
4.4. Infrarotlicht

Der IR-Emitter kann mit einer Knopfzelle und Isolierband an einem Carrera-Wagen befestigt werden, um das System im vorgesehenen Szenario zu testen. Der IR-Emitter sollte an einer Seite befestigt sein, von der aus das Licht von der Kamera vollständig erfasst werden kann.

Zu generellen Testzwecken kann der IR-Emitter auch auf einer Steckplatine befestigt werden. Die Stromzufuhr sollte dann über Verbindungskabel mit einer Batterie geregelt werden. Die Steckplatine ist zum Testen der Funktionalitäten handlicher als der Carrera-Wagen.



IR-Emitter auf Steckplatine



Vollständige Konstruktion

5. Umsetzung

5.1 Benutzte Mittel

Das Programm wurde vor der Implementierung auf dem Raspberry Pi in der “CLion” IDE in C++ 11 entwickelt und über Git versioniert. Auf dem Gerät ist das Betriebssystem “Raspbian Jessie” installiert.

Die Ansteuerung der Servomotoren wird durch die “PiGPIO” Bibliothek realisiert. Über diese werden vom Programm übergebene Gradzahlen in eine Pulsweite umgewandelt, die an die Motoren weitergegeben wird. Die Bibliothek verhindert weiterhin das Stottern der Motoren und ermöglicht eine flüssige Bewegung.

Die Bildverarbeitung wird mit der “OpenCV” Bibliothek umgesetzt. Das aufgenommene Kamerabild muss für die Erkennung gefiltert und auf bestimmte Eigenschaften untersucht werden. Algorithmen dafür liefert die OpenCV Bibliothek.

5.2. Klassen

Das Programm besteht aus 3 Klassen.

- main
- SearchCART
- Servo

5.2.1. main Klasse

Die Klasse “main” ist die Hauptklasse des Programms. Sie initialisiert die anderen beiden Klassen und startet die Schleife, in der das Programm läuft.

5.2.2. SearchCART Klasse

“SearchCART” übernimmt die Verarbeitung der Kamerabilder und die Identifizierung der Eigenschaften des Bildes.

5.2.3. Servo Klasse

Die “Servo” Klasse stellt eine Verbindung zu den Motoren her und übernimmt die Umrechnung der Winkel in Pulsweiten, die die Motoren annehmen können.

5.3. Programmdurchlauf

Das Programm beginnt in der “main” Klasse. Diese initialisiert zunächst die “Servo” Klasse so wie die “SearchCART” Klasse. Dadurch wird in der “Servo” Klasse die Position des Servos auf die Ausgangsposition, (X,Y), gesetzt. Danach beginnt das Programm eine endlose Schleife durch `while (true)`.

In der Schleife wird zunächst von der “SearchCART” Klasse erfragt, ob ein “Zielobjekt”, also ein Kreis gefunden wurde. Die Funktion `SearchCART::isNoTarget()` liefert einen boolean, der angibt, ob ein Kreis im letzten Frame erkannt wurde oder nicht. Nachdem diese Funktion sechzig mal `true` zurückgibt, wird der Servo durch `Servo::reset()` wieder auf die Ausgangsposition zurückgesetzt.

Nach der `isNoTarget()` Überprüfung wird durch `SearchCART::loader(int index)` der aktuelle Frame auf “Zielobjekte” untersucht. In dieser Funktion wird zunächst durch `capture >>` ein aktuelles Frame von der Kamera geholt. Durch `cvtColor()` und `GaussianBlur()` wird das Bild in Graustufen konvertiert und verwaschen, damit nicht alles als Kreis erkannt wird. Über `HoughCircles()` wird nach Kreisen im Bild gesucht und die Position des Mittelpunktes von der Funktion zurückgegeben. Wenn kein Kreis gefunden wird, wird die mittlere Position des Bildes zurückgegeben.

Nachdem ein Kreis identifiziert wurde, wird die Position des Kreises an die main Klasse weitergegeben. Die aktuelle Position des Kreises im Kamerabild wird gegen bestimmte Schwellwerte durch mehrere `if/else` Abfragen abgeglichen. Basierend auf der Position im Verhältnis zu den Schwellwerten wird dann in `moveDegreeX` und `moveDegreeY` ein Wert festgesetzt, der später den Servomotoren durch `Servo::degreeToPw(int degreeX, int degreeY)` übergeben wird.

In der `degreeToPw(int degreeX, int degreeY)` Methode werden die übergebenen Gradzahlen in die benötigte Pulsweite umgerechnet, sodass der Servo sie benutzen kann. Sobald die Werte berechnet sind, werden sie für den jeweiligen Servomotor durch `Servo::moveServo(int servo, int pw)` übergeben. `int servo` identifiziert hierbei welcher Servo angesteuert werden, also ob eine vertikale oder horizontale Bewegung ausgeführt werden soll. Über die Funktion `gpioServo(int servo, int pw)` aus der “PiGPIO” Bibliothek wird dann die Bewegung an die Servomotoren weitergeleitet.

Danach beginnt die Schleife in der main Klasse wieder von vorne.

5.4 Anmerkung zu SearchCART

In einem früheren Versuch wurde ausprobiert, die Erkennung mit Hilfe eines Thresholdes durchzuführen. Dabei wurden alle Bereiche des Bildes, welche heller als der Threshold sind auf die maximale Helligkeit gesetzt. Alle Teile die dunkler waren, wurden auf Schwarz gesetzt. Allerdings führte dies zu verstreuten Bereichen, die zu hell waren. In diesem Versuch wurde der Mittelpunkt der Fläche mit weißen Bereichen als Ziel gesetzt. Die Kreiserkennung mit Hough hat beim Threshold nicht zuverlässig funktioniert.

6. Anleitung

Um den Prototypen zu testen, muss zunächst der Aufbau auf Vollständigkeit und Korrektheit überprüft werden. Der Raspberry Pi sollte dann mit Maus, Tastatur und Bildschirm verbunden werden, damit das Programm im Betriebssystem aufgerufen werden kann.

Bevor das Programm ausgeführt werden kann, muss das Kernel-Modul für die Kamera im System aktiviert werden. Dafür muss das LX Terminal aufgerufen und der folgende Befehl eingegeben werden:

```
cd Documents/cart/src/
./mod.sh
```

Nachdem dieser Schritt ausgeführt wurde, muss das Programm selbst gestartet werden. Dafür muss wieder die Konsole aufgerufen und dieser Befehl eingegeben werden:

```
sudo ./CART
```

Sobald sich der Kamerakopf auf die Anfangsposition bewegt, kann der Testdurchlauf beginnen. Damit die Erkennung des Infrarotlichts als Kreis besser funktioniert sollte der Raum abgedunkelt werden.
Der Infrarot Emitter kann nun am Carerra-Wagen befestigt werden. Um zu testen, ob der Emitter richtig funktioniert, kann das Livebild, welches über den Bildschirm ausgegeben wird, benutzt werden. Wenn der Infrarot Emitter korrekt funktioniert, verfolgt der Kamerakopf das Licht.

7. Fazit

Der Prototyp ist funktionsfähig. In einem abgedunkelten Raum kann der Kamerakopf ein sich langsam bewegendes Infrarotlicht verfolgen. Ein Carrera-Wagen kann ebenfalls mit dem System verfolgt werden, wenn es sich bei langsamer Geschwindigkeit in kurzen Stößen über die Strecke bewegt.

Leider funktioniert die Verfolgung weniger optimal bei hohen Geschwindigkeiten des Infrarotlichts. Dies liegt zum einen an der Bewegungsunschärfe in einem Frame, was sich vermutlich durch eine Erhöhung der aufgenommenen Frames pro Sekunde verbessern könnte.

Die Bewegungsunschärfe verwischt den Kreis, den das Infrarotlicht ausgibt. Eine weitere Ausbaumöglichkeit wäre also die Verbesserung der Bilderkennung durch einen Algorithmus, der das Infrarotlicht besser erkennt und sich dabei nicht auf Kreisformen verlässt. Dies würde außerdem in einem weniger gut abgedunkelten Raum die Anzahl der falschen Zielerkennungen im Bild verringern.

Des Weiteren wäre eine Verbesserung des Bilderkennungsalgorithmus durch das “Trainieren” des Programms mit “Haar Cascades” [6] möglich. Dieses OpenCV Konzept ermöglicht automatisches Erkennen eines Objektes an seinen Eigenschaften. Mit “Haar Cascades” wurde bereits die Erkennung von Automobilen im Verkehr ermöglicht [7], daher liegt die Erkennung eines Carrera-Wagens nicht fern.

Anhang

main.h

```
#ifndef _CART_MAIN_
#define _CART_MAIN_

#include "SearchCART.h"
#include "Servo.h"
#include <iostream>

using namespace std;

int main();

#endif  // _CART_MAIN_
```

main.cpp

```
#include "main.h"

int main() {
   SearchCART search;
   Servo servo;

   int resolutionX = search.getSize()[0];
   int resolutionY = search.getSize()[1];
   int sectorX = resolutionX / 5;
   int sectorY = resolutionY / 5;
   int innerLeftUpX = sectorX * 2;
   int innerLeftUpY = sectorY * 2;
   int innerLeftDownY = sectorY * 3;
   int innerRightUpX = sectorX * 3;
   int outerLeftUpX = sectorX;
   int outerLeftUpY = sectorY;
   int outerLeftDownY = sectorY * 4;
   int outerRightUpX = sectorX * 4;
   int currentDegreeX;
   int moveDegreeX;
   int currentDegreeY;
   int moveDegreeY;

   // Ausgabe der Größe des Videos
   cout << "width:  "
        << resolutionX << endl
        << "height: "
        << resolutionY << endl
        << "fps: "
        << search.getFPS() << endl;

   //Initialisiert die Servos auf die Anfangsposition
```

```
    if (servo.initialiseServo()) {
        currentDegreeX = 0;
        currentDegreeY = 30;
    }

    // schrittweise das Video durchlaufen und die Servos korrigieren.
    int i = 0;
    int noTarget = 0;
    while (true) {

        if (search.isNoTarget()) {
            noTarget++;
        } else {
            noTarget = 0;
        }

        if (noTarget > 60) {
            servo.reset();
            currentDegreeX = 0;
            currentDegreeY = 30;
            noTarget = 0;
        }

        if (true) {
            vector<int> position = search.loader(i);

//          cout << "X: "
//               << position[0]
//               << " Y: "
//               << position[1] << endl;

            double tPosX = position[0];
            double tPosY = position[1];

            cout << "target position is " << tPosX << " | " << tPosY << endl;

            //Überprüft die x-Position.
            if (tPosX > innerLeftUpX) {
                if (tPosX < innerRightUpX) {
                    //Servo muss nicht in x-Richtung bewegt werden.
                    moveDegreeX = 0;
                } else {
                    if (tPosX < outerRightUpX) {
                        //xSlowPositive
                        //Servo muss langsam in x-Richtung bewegt werden.
                        moveDegreeX = -2;
                    } else {
                        //xFastPositive
                        //Servo muss schnell in x-Richtung bewegt werden.
                        moveDegreeX = -4;
                    }
                }
            } else {
                if (tPosX > outerLeftUpX) {
```

```
                //xSlowNegative
                //Servo muss langsam in x-Richtung bewegt werden.
                moveDegreeX = 2;
            } else {
                //xFastNegative
                //Servo muss schnell in x-Richtung bewegt werden.
                moveDegreeX = 4;
            }
        }

        //checks position of y.
        if (tPosY > innerLeftUpY) {
            if (tPosY < innerLeftDownY) {
                //YOK
                moveDegreeY = 0;
            } else {
                if (tPosY < outerLeftDownY) {
                    //ySlowNegative
                    //Servo muss langsam in y-Richtung bewegt werden.
                    moveDegreeY = 1;
                } else {
                    //yFastNegative
                    //Servo muss schnell in y-Richtung bewegt werden.
                    moveDegreeY = 2;
                }
            }
        } else {
            if (tPosY > outerLeftUpY) {
                //ySlowPositive
                //Servo muss langsam in y-Richtung bewegt werden.
                moveDegreeY = -1;
            } else {
                //yFastPositive
                //Servo muss schnell in y-Richtung bewegt werden.
                moveDegreeY = -2;
            }
        }
        //Wenn die Bewegung möglich ist, wird der aktuelle Winkel aktualisiert.
        if (servo.degreeToPw(currentDegreeX + moveDegreeX,
                             currentDegreeY + moveDegreeY) != 0) {
            currentDegreeX = currentDegreeX + moveDegreeX;
            currentDegreeY = currentDegreeY + moveDegreeY;
        }

        cout << endl;
    }

    i++;
  }
}
```

Servo.h

```
#ifndef CART_MOVE_TEST_H
#define CART_MOVE_TEST_H


#include <iostream>
#include <pigpio.h>

using namespace std;

class Servo
{
private:
   void moveServo(int servo, int pw);
public:
   int initialiseServo();
   int degreeToPw(int degreeX, int degreeY);
   void reset();
};

#endif //CART_MOVE_TEST_H
```

Servo.cpp

```
#include "Servo.h"

// Maximale Frequenzen für den Arbeitsbereich der Servos
int maxPwX = 1900;
int minPwX = 1100;
int maxPwY = 1900;
int minPwY = 1100;

// GPIO Adressen der Servos
int SERVO_X = 20;
int SERVO_Y = 26;

/** Initialisiert pigpio und setzt die Servos auf die Ausgangsposition.
* @return int mit dem Erfolg.
*/
int Servo::initialiseServo() {
   if (gpioInitialise() < 0) {
        cout << "ERROR initializing!" << endl;
        return 0;
   } else {
        this->reset();
        return 1;
   }
}

void Servo::reset() {
   gpioServo(SERVO_X, minPwX);
```

```
    gpioServo(SERVO_Y, minPwY + (maxPwY - minPwY) / 2);
    time_sleep(3);
}

/** Wandelt die Gradzahlen in Pulsweite um.
* @param degreeX Grad der Veränderung in x-Richtung.
* @param degreeY Grad der Veränderung in y-Richtung.
* @return ob die Bewegung möglich ist.
*/
int Servo::degreeToPw(int degreeX, int degreeY) {
    cout << "degreeX: "
         << degreeX
         << " degreeY: "
         << degreeY << endl;
    int rangeX = maxPwX - minPwX;
    int oneDegX = rangeX / 144;
    int rangeY = maxPwY - minPwY;
    int oneDegY = rangeY / 48;
    int resultX = minPwX + (oneDegX * degreeX);
    int resultY = minPwY + (oneDegY * degreeY);
    //Verhindert dass der Servo außerhalb seiner Grenzen fährt.
    if (resultX >= minPwX && resultX <= maxPwX && resultY >= minPwY && resultY <=
maxPwY){
        moveServo(SERVO_X, resultX);

    //if (resultY >= minPwY && resultY <= maxPwY)
        moveServo(SERVO_Y, resultY);
        return 1;
    } else {
        return 0;
    }
}

/** Bewegt den Servo auf die gegebene Pulsweite.
* @param servo Der Servo der angesteuert wird.
* @param pw Die Pulsweite die angelegt wird.
*/
void Servo::moveServo(int servo, int pw) {
    gpioServo(servo, pw);
    cout << "Servo: "
         << servo
         << " at PW: "
         << pw << endl;
}
```

SearchCART.h

```
#ifndef _CART_SEARCH_
#define _CART_SEARCH_

#include <iostream>
#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/core/core.hpp>
#include <sstream>

using namespace cv;
using namespace std;

class SearchCART
{
private:
   bool noTarget = false;
   bool connected = false;
   int initVideo();
   VideoCapture capture;
public:
   SearchCART();
   ~SearchCART();
   vector<int> loader(int i);
   vector<int> getSize();
   double getFPS();
   bool isConnected();
   bool isNoTarget();
};

#endif  // _CART_SEARCH_
```

SearchCART.cpp

```
#include "SearchCART.h"

SearchCART::SearchCART() {
   cout << "con" << endl;
   initVideo();
}

SearchCART::~SearchCART() {
   cout << "des" << endl;
   capture.release();
}

/**
* Initialisiert den Stream.
* @return 0 wenn ok, -1 wenn Stream nicht offen ist.
*/
int SearchCART::initVideo() {
```

```
    // Pfad zum Testvideo (build wird nicht im src-Verzeichnis abgelegt)
    string path = "../res/testvideo/";
    string filename = "test_v1.mov";
    string fullFilename = path + filename;

    cout << fullFilename << endl;

    // Öffnen des Streams
    capture.open(0);
    capture.set(CAP_PROP_FPS, 30);
    if ( !capture.isOpened()) {
        this->connected = true;
        return -1;
    }
    else
        cout << "STREAM IS OPEN" << endl;
    return 0;
}

/**
* Getter für die Dimension des Streams in Pixeln.
* @return int Vector {Breite, Höhe}.
*/
vector<int> SearchCART::getSize() {
    int width = (int) capture.get(CAP_PROP_FRAME_WIDTH);
    int height = (int) capture.get(CAP_PROP_FRAME_HEIGHT);

    vector<int> retVal {width, height};
    retVal.shrink_to_fit();

    return retVal;
}

/**
* Getter für die Bilder pro Sekunde.
* @return double Bilder pro Sekunde.
*/
double SearchCART::getFPS() {
    return capture.get(CAP_PROP_FPS);
}

/** Getter für den Status der Verbindung zur Kamera.
* @return bool true wenn die Kamera verbunden ist.
*/
bool SearchCART::isConnected() {
    return this->connected;
}

bool SearchCART::isNoTarget() {
    return this->noTarget;
}

/**
* Lädt die Koordinaten des Punktes aus dem aktuellen Bild.
```

```
* @param index Der Index der Schleife für die Speicherung der Sequenz.
* @return int Vector mit den Koordinaten des Ziels.
* Wenn kein Ziel erkannt wurde, wird der Mittelpunkt des Bildes zurück gegeben. {x,
y}
*/
vector<int> SearchCART::loader(int index) {
    // Das aktuelle Frame laden
    Mat captureRGB;
    capture >> captureRGB;

    // In Graustufen konvertieren
    Mat captureGray;
    cvtColor(captureRGB, captureGray, CV_RGB2GRAY);

    // Schwellwert für Graustufen (alles unter dem Schwellwert soll schwarz werden)
//    int threshold_value = 200;

//    Mat captureThres;
//    threshold(captureGray, captureThres, threshold_value, 255, 0);

    // Auslesen der weißen Pixel
//    Mat whitePixelCoord;
//    findNonZero(captureThres, whitePixelCoord);

//    // Berechnen des Mittelpunktes der Ansammlung von weißen Pixeln
//    int startX = whitePixelCoord.row(0).at<int>(0,0);
//    int startY = whitePixelCoord.row(0).at<int>(0,1);
//
//    int stopX = whitePixelCoord.row(whitePixelCoord.rows-1).at<int>(0,0);
//    int stopY = whitePixelCoord.row(whitePixelCoord.rows-1).at<int>(0,1);
//
//    int midX = startX + ((stopX - startX) / 2);
//    int midY = startY + ((stopY - startY) / 2);

//    vector<int> ballPosition(midX, midY);

    // Speichert die möglichen Lichtquellen
    vector<Vec3f> possibleLights;

    // Blur um Irritationen auszuschließen
    GaussianBlur(captureGray, captureGray, Size(9, 9), 2, 2);
    // Finden von Kreisen im Bild
    HoughCircles(captureGray, possibleLights, CV_HOUGH_GRADIENT, 1,
captureGray.rows, 180, 20, 2, 50);

    vector<int> targetCoord(2);

    // Hole den ersten Kreis sofern welche vorhanden sind
    // Ansonsten setze den Mittelpunkt des Bildes als Ziel
    if (possibleLights.capacity() > 0) {
        this->noTarget = false;
        int targetX = (int) possibleLights[0].val[0];
        int targetY = (int) possibleLights[0].val[1];
        targetCoord = {targetX, targetY};
```

```
    } else {
        cout << "NO TARGET FOUND!" << endl;
        this->noTarget = true;
        int midWidth = this->getSize()[0] / 2;
        int midHeight = this->getSize()[1] / 2;
        targetCoord = {midWidth, midHeight};
    }

    // Für zu speichernde Bilder: Zeichnen der Kreise in das Farbbbild
    for( size_t i = 0; i < possibleLights.size(); i++ )
    {
        Point center(cvRound(possibleLights[i][0]), cvRound(possibleLights[i][1]));
        int radius = cvRound(possibleLights[i][2]);
        // circle center
        circle( captureRGB, center, 3, Scalar(0,255,0), -1, 8, 0 );
        // circle outline
        circle( captureRGB, center, radius, Scalar(0,0,255), 3, 8, 0 );
    }

    // print white pixel positions
//    if (index == 0) {
//        //cout << "white pixels at = "<< endl << " "  << whitePixelCoord << endl
<< endl;
//    }

    // schreibe die Einzelbilder auf die Platte
    ostringstream oss;
    oss << index;
    string savepath = "img/image.";
    savepath += oss.str();
    savepath += ".jpg";
//    imwrite(savepath, captureRGB);

    // Show Liveview;
    imshow("Live@CART", captureRGB);
    waitKey(1);

    return targetCoord;
}
```

Quellen

[1] Krishna, M. Poddar, Giridhar M K, A. S. Prabhu and Umadevi V, "Automated traffic monitoring system using computer vision," *2016 International Conference on ICT in Business Industry & Government (ICTBIG)*, Indore, 2016, pp. 1-5.
[2] https://soloshot.com
[3] https://shop.pimoroni.com/products/raspberry-pi-3
[4] https://www.conrad.de/de/pan-tilt-roll-mechanismus-fat-shark-1196364.html
[5] https://www.conrad.de/de/raspberry-pi-kamera-gehaeusemodul-camera-v2-8mp-ir-raspberry-pi-raspberry-pi-2-b-raspberry-pi-3-b-raspberry-pi-a-1455913.html
[6] http://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html
[7] https://www.youtube.com/watch?v=c4LobbqeKZc